AFRL-IF-WP-TM-2002-1562

SYSTEMS LEVEL DESIGN LANGUAGE (SLDL) PROJECT

David L. Barton

Titan Corporation
Civil Government Systems Group
1593 Spring Hill Road, Suite 700
Vienna, VA 22182-2249

JANUARY 2002

Final Report for 29 January 1999 – 02 January 2002

20020926 085

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
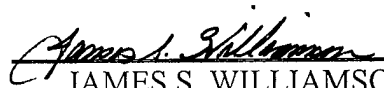WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT HAS BEEN REVIEWED BY THE OFFICE OF PUBLIC AFFAIRS (ASC/PA) AND IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.

MICHAEL T. MILLS
Project Engineer

JAMES S. WILLIAMSON, Chief
Embedded Info Sys Engineering
Information Technology Division
Information Directorate

for EUGENE BLACKBURN, Chief
Information Technology Division
Information Directorate

This report is published in the interest of scientific and technical information exchange and does not constitute approval or disapproval of its ideas or findings.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE

1. REPORT DATE (DD-MM-YY)	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2002	Final	01/29/1999 – 01/02/2002

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
SYSTEMS LEVEL DESIGN LANGUAGE (SLDL) PROJECT	F33615-98-C-1350
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62204F

6. AUTHOR(S)	5d. PROJECT NUMBER
David L. Barton	6096
	5e. TASK NUMBER 40
	5f. WORK UNIT NUMBER 3A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Titan Corporation Civil Government Systems Group 1593 Spring Hill Road, Suite 700 Vienna, VA 22182-2249	

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY ACRONYM(S)
Information Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB, OH 45433-7334	AFRL/IFTA
	11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-IF-WP-TM-2002-1562

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The purpose of this final report is to document the development process that resulted in a System Level Design Language (SLDL) Specification and associated Parser. The resulting language, called Rosetta, and prototype tools and updates are available to the public on web sites http://www.sldl.org or http://www.ittc.ku.edu/Projects/rosetta at the time of this report.

14. ABSTRACT

The objective of the Systems Level Design Language (SLDL) Program was to provide a new mechanism for systems design that will allow a number of models, created in different design domains (or specialties), to share information and to create a unified systems design environment. The resulting language can represent large system descriptions and assist system developers to handle the complexity inherent in designing and understanding such systems. As commercial tools are developed to interact with and use the SLDL language specification, currently known as Rosetta, system developers will be better able to catch design flaws much earlier in the design process, resulting in significant savings in rework and associated development cost.

Although this contract funded only 2 years of a planned 3-year effort, it succeeded in providing the momentum which produced results that could be picked up by other efforts. Dedicated efforts from the SLDL committee formed by the Electronic Design Automation (EDA) Industry Council and the language and tool developers resulted in continued progress heading toward an inevitable improvement in system level design capabilities applicable to both commercial and military systems.

15. SUBJECT TERMS

SLDL, Rosetta, Systems Level Design Language

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	SAR	18	Michael T. Mills 19b. TELEPHONE NUMBER (Include Area Code) (937) 255-6548 x3583

HES&S 31-15093-1

Final Report for the SLDL Project

David L. Barton
Titan Corp., Civil Government Systems Group
1593 Spring Hill Rd.
Vienna, VA 22182
(703) 827-2606

January 2, 2002

1 Introduction

The industrial need for new systems design mechanisms is clear. Moore's Law leads to an inevitable conclusion that our production capability will, sooner or later, outstrip our design capability. With the advent of Systems on Chip (SoC), this time has come. The EDA (Electronic Design Automation) Industry Council created the SLDL committee to do just this: create a new mechanism for systems design, that will allow a number of models, created in different design domains, to share information and thus to create a unified systems design environment.

At the center of this has to be some means of describing the entire meaning, or semantics, of a systems description that stretches over many engineering domains, consists of hundreds or thousands of models in different languages and created using different tools, and that uses a large variety of analysis mechanisms and software to assure that the system meets its specification. Such a language has a tremendous task, as it must not only represent this large description, but must assist the user in conquering the complexity inherent in designing and understanding such a system.

Government requirements of electronic systems are in many ways identical to those of industry; however, in some ways they are unique. To ensure that the needs of military systems were taken into account in any results coming out of the SLDL committee, the Air Force decided to commit some funds to support that development. In this way, the Air Force both assured the timely and appropriate development of any solutions to the SLDL effort, and assured that

1

government requirements would be folded into the SLDL process.

This report will give a brief history of the SLDL project, a summary of the benefits of the project, and a review of the current status of the Rosetta language and tools for that language. The SLDL effort is ongoing, and the eventual road for Rosetta is towards standardization, first through Accellera, and later through the IEEE. The language is on track to attain that goal.

The reader should recall that during this project the last third of the funding was canceled. Funding disruptions and cuts are a fact of life in government contracts, but there cannot help but be some effect. We believe we have met the goals of the original program to a large extent; however, the final result is not as developed as we would have wished. We will address this further in the lessons learned section below.

2 History of the SLDL Committee

The SLDL committee was formed as a technical committee of the EDA Industry Council. Steve Schulz chaired the first meeting, and David Barton was appointed chair after that meeting. A series of information gathering meetings were organized in Silicon Valley and Europe. As a result of these meetings, a number of subcommittees were formed. These subcommittees met to create requirements, and created a number of separate sets of requirements. These were integrated into the overall SLDL requirements document.

In addition to the requirements document, the committee gathered a number of "representative examples" that represent the current state of the art in systems design. These examples were surprisingly short, and established that systems designers often create a design consisting of a block diagram, a number of parameters and not much else, handing the rest of the design task off to individual members of the design team.

After creating the requirements and the representative examples, the SLDL committee moved on to invite industry to present tools and current practices that could be judged against the current set of examples. This resulted in the definition of a number of "holes", or requirements in the document that do not seem to be covered by present industry tools.

As a result of these holes, the following committee structure was created:

- A Language Committee, whose task was to create a Systems Level Design Language (SLDL) specifically oriented to describing constraints and parameters on systems and on system components.

- A Semantics committee, whose task was to create a set of conceptual semantic definitions by which languages and systems tools could be compared. This will allow the industry to compare "apples to apples" when examining systems level design tools.

2

About this time, the SLDL committee found that the EDA Industrial Council was not meeting the committee's needs with respect to support and publicity. The SLDL committee therefore moved under VHDL International (VI), which was interested in moving its activities into the systems design arena. Under the auspices of VI, the SLDL committee held a number of explanatory meetings and continued technical work on the language development.

As the language design neared completion, VI decided to merge with a related organization, Open Verilog International (OVI) to form Accellera. The SLDL committee was, for a time, a technical committee under Accellera. Recently, in a streamlining move, the SLDL committee has been eliminated and the Language committee (now the Rosetta committee) and the Systems and Semantics committee are now technical committees in their own right.

In its present form, the Rosetta and Systems and Semantics committees have the full support of Accellera, which is a standardization group. This will allow the Rosetta and Semantics efforts to continue, and to proceed directly to standardization, without the mechanism and long balloting periods of the IEEE. When the standard progresses to a level of stability, it will be turned over to the IEEE for a more broad constituency.

3 Scope

The scope of the SLDL committee is given in its mission statement. That statement is:

"The mission of the SLDL committee is to support the creation and/or standardization of a language or representation, or set of languages or representations, that will enable engineers to describe single and multi-chip silicon-based embedded systems to any desired degree of detail and that will allow engineers to verify and/or validate those systems with appropriate tools and techniques. The languages or representations will address systems under design and their operating environment, including software, mixed-signal, optical, and micro-electronic mechanical characteristics."

This mission statement is exceptionally broad. This broad scope sets the SLDL effort apart from other systems design languages and efforts. The result is that the committee, and the resulting languages and notations, must take an extremely high level view of a system, and allow multiple domains and kinds of engineering discipline into a single systems description. This is reflected in the requirements for the language.

4 Requirements

One result of the SLDL committee work was a full set of requirements for the language that would be produced. This language was called Rosetta, after the

Rosetta stone. The full requirements document may be found on the committee web site, http://www.inmet.com/sldl, but we summarize the requirements here:

Scope: Rosetta must stretch across multiple application domains and disciplines: software, digital, analog, microwave/RF, mechanical (MEMs), optical, and anything else that may be placed on a silicon chip. However, nothing limits Rosetta to silicon.

Areas of Concern: Rosetta must specifically cover requirements and constraints on the system, as well as the system functionality (what the system does). Of great concern are limits within which the system must operate (environmental, power consumption, and so on).

Structure: Rosetta may have emphasis on other areas, but because systems designers still work with block diagrams, structure is important within the language. This includes how the system is put together and what it looks like (schematics versus masks).

Requirements and Constraints: Rosetta must be extremely flexible in its ability to specify requirements and constraints on the system. Specifically, it must:

- Specify partial descriptions, i.e. descriptions with only partial information available.
- Specify and combine different abstraction levels in a hierarchical decomposition.
- Specify the values, or range, of parameters and their meaning. This means the mathematics of such concepts as decibel (and many others).
- Divide the "responsibility" for requirements between different components and "views" of the system.
- Track the satisfaction of the requirement.
- Define metrics, record measurements, and relate them to constraints and requirements.

Implementation Independence: Rosetta must be able to specify interfaces between subsystems / components in different implementation domains. This means that subsystem interfaces must be free of any kind of domain or implementation bias (except that imposed by the requirement itself).

Tools: A variety of tools must be able to operate on a Rosetta specification. Such a variety means that Rosetta is not defined as VHDL or C is, as an abstract implementation. Rosetta will be defined formally (see below). The tools include at least (but are not limited to):

- The execution, or simulation, of the specification;

- The easy entry and manipulation of requirements and descriptions;
- Knowledge bases of requirements;
- The use of templates to fill in system requirements;
- The ability to customize the language for different implementation domains (i.e., to use tools and notations from different domains).

Interfaces There are a variety of concerns in the area of interfaces between components and between views. These include:

- Users need the capability to map system level behavioral requirements to an implementable description (a description that could be further analyzed and/or synthesized).
- Users need the capability to reflect system level constraint requirements into the implementable component level domain.
- The notation must provide the ability to pass information bottom up.
- Users need the capability to map system level interface protocols into known or new ways of communication at the interface level.
- We must have the capability to describe negotiation mechanisms between requirements and implementation. Implementors may fail to meet their requirements, and when this happens we need a mechanism to figure out how to change requirements across the system components so that the overall system meets its specification.
- We must have mechanisms to allow the user to protect proprietary information in his models.
- Rosetta must be able to express performance models for standard products.
- The user must be able to establish a structured mapping between an event at the systems level and an event, or a set of events, at the component level.
- The user must be able to use various abstraction mechanisms in his systems description, including black box specification and abstract component model specification.

Formal Semantics: The formal semantic definition of Rosetta must have certain characteristics, including:

- Sufficient power;
- Domain theories;
- Partial specifications;
- Composable specifications;
- Description characterizations;

5

- Extensibility;
- Different component notations;
- Views.

As can be seen, this is a formidable set of requirements, and it requires a formidable language to respond to it. We include a brief description of the language in the following section. More information may be found on the Rosetta web site, http://www.sldl.org.

5 The Rosetta Language

In order to meet the requirements imposed on it by the SLDL committee, Rosetta provides language support for defining and combining models from multiple domains using multiple domain semantics. In addition, Rosetta provides support for modeling and analysis at levels of abstraction much higher than current RTL based languages. Its semantics are formally defined and it is highly extensible to support adaptation to emerging systems. We will look at the various parts of the Rosetta language in this section.

5.1 Facets and Models

The Rosetta design methodology is based on the facet modeling concept. A facet is a model of a component or system that provides information specific to a domain of interest. To support heterogeneity in designs, each facet may use a different domain model to provide domain specific vocabulary and semantics. Facets are written to define various system aspects and are then assembled to provide complete models of components, component aggregations and systems.

The definition of facets is achieved by directly defining model properties or by combining previously defined facets. The former technique allows users to choose a specification domain and specify properties that must hold in that domain. The latter technique allows users to select several models and compose a system model that consists of elements from each facet domain. The abstract semantics of Rosetta is based upon this specification and combination of models representing information from various design domains.

The syntax of Rosetta facets is designed to be familiar to engineers using existing hardware description languages. Figure 1 shows an annotated specification of the functional aspects of Schmidt trigger. Although the specification is quite simple, it demonstrates many aspects of the facet specification concept.

Figure 1 - Rosetta specification of a Schmidt trigger.

As with traditional hardware description techniques, the trigger specification opens by naming the model (schmidt) and defining a parameter list. A local variable (s) is also defined to describe the facet state and the begin keyword

```
facet schmidt (input: in real; output:: out bit) is

  s:: bit;

begin continuous

  t1: s' =  if s = 0 then
                if input > 0.7 then 1 else 0
            else
                if input < 0.4 then 0 else 1;
  t2: output@t+10ns = s;
end schmidt;
```

Figure 1: Schmidt Trigger in Rosetta

opens the specification body. Following the begin keyword a domain (continuous) for the specification is identified. Finally, two terms (t1 and t2) are defined that describe the behavior of the trigger. What makes the facet model different than traditional HDL approaches is the identification of the domain. A primarily difficulty in addressing the systems-on-chip problem is combining in formation from multiple domains in a single design activity. The domain allows this by allowing each model to reference a base set of concepts that support definition of information in one particular design area. In the trigger specification, the continuous domain provides the concepts of time, instantaneous change, ordering and state. Such concepts are important to continuous time modeling, but may not be important when modeling power consumption or area. Such models reference the constraints domain for defining such information.

The constraints domain provides the trigger specification with concepts of power, heat dissipation, area and other non-functional design requirements. Unlike the continuous domain, the constraints domain has no concept of state or time as such concepts are not needed to describe current performance information.

To define a Schmidt trigger that satisfies both requirements sets, facet composition operators are provided. The user can use facet conjunction to define a new Schmidt trigger facet that combines both specification models and asserts that both must hold simultaneously. The declaration defines a new facet called trigger and then asserts that the new facet is the conjunction of the trigger requirements and performance models. Effectively, if a trigger is examined, it can be viewed from either a functional or requirements perspective.

The Rosetta semantic model, i.e. defining and combining models, is important as it reflects how systems engineers currently tackle their problems. The syntax makes the semantics approachable, however the real contribution of Rosetta is the semantics of designing and composing of component models. Through the use of domains, users are provided with design vocabulary specific to their

domain of expertise rather than forcing the domain expert to work in a language that is to general or otherwise unsuitable for their needs.

5.2   Systems and Heterogeneity

Heterogeneity in systems design not only emerges when defining multiple facets of the same system, but also when describing systems structurally by combining components. VHDL provides a structural definition capability that is mimicked in the Rosetta semantics using a concept called re-labeling. When one facet is included in another, an instance of that facet is created by re-labeling or renaming the facet. Facet parameters are used as ports and channel data types used to model communication between components. Systems level components can be included and combined in an abstract specification. For example, an analog to digital converter, modulator and amplifier are used in series to produce a signal suitable for output to an antenna. Both a functional model and power consumption model can be created. When composed, the two facets define a systems level transmitter with a mechanism for calculating power consumption from constituent components. Further, the semantics of facet inclusion maps each power constraint facet to its associated functional facet. This allows power constraints to be automatically mapped to corresponding functional constraints.

Like single components, Rosetta systems may include component models from multiple design domains. Thus, Rosetta provides a mechanism for defining and understanding systems comprised of components from analog, digital, mechanical and optical domains in the same semantic framework. Furthermore, users may define new specification domains to extend Rosetta to address new domains and new modeling techniques for existing domains. Such a capability is extremely important for the future of any potential systems design language.

5.3   Domain Interaction

Unfortunately, it is not sufficient to simply model domain specific information in isolation. Cross domain interaction is the root cause of many systems failures and difficult design problems. Systems level design requires understanding the collective behavior of interconnected components from different design domains, not simply the component level behaviors. Further, interaction also occurs between different models at the component level. Rosetta provides methodologies for explicitly modeling and evaluating such interactions by defining how definitions from individual domains affect each other.

There are two fundamental approaches for modeling interaction between design domains in current design practice. The first is to choose a single domain for representing all system models. Analysis tools from that domain are then used to model entire systems. The problem with this approach is that the modeling domain always favors one domain over all others. Designers or tool builders must force information represented quite naturally in one domain into a potentially

8

less suitable representation. The net result is a model that cannot be analyzed or a systems model that ignores certain domains.

The second approach to addressing model interaction is to rely on human observation to detect and analyze interactions between domains. Models are developed and maintained using separate representations and semantics. Human designers are required to constantly observe and predict interactions. This approach is the most common in today's environment and will most probably always exist to some extent. However, Rosetta attempts to provide semantic support for this activity by providing a mechanism for specifying and thus analyzing domain interactions.

Rosetta provides the special domain and interaction semantic constructs for defining domains and domain interactions respectively. The domain construct simply provides a mechanism for identifying a facet as a domain model. Syntax is provided in Rosetta to allow users to identify and define domains quickly, but their semantics is the same as a facet. When a facet includes a domain, it extends that domain to represent a specific component or system model. When facets using domains involved in an interaction specification are included in a definition, the interaction specification is automatically included.

Like a domain, an interaction is also a facet. Using Rosetta's reflection capabilities, an interaction defines when specifications in one domain imply specifications in another. An interaction is a facet that accepts two domains as parameters. It then defines when information in one domain is pertinent in another.

Rosetta currently provides semantics for mathematical specifications, state-based (finite and infinite), discrete time, and continuous time specification. Support for specification of testing requirements, constraints and mechanical systems are currently being developed. Interactions are defined for many supported domains with others under development.

It is important to note that Rosetta's semantics are defined independently from its concrete syntax. Although a facet does have a defined syntactic representation, its semantics were defined initially and syntax developed to address usability and readability issues. Other syntactic representations are not precluded and are being encouraged.

6 Benefits of this Effort

This contract has had a great deal of influence on the Rosetta language and effort. First and foremost, it has assured a sustained level of effort that resulted in a stable language definition. Second, it allowed government requirements to be included in the Rosetta development process.

In addition to the language development itself, this contract has allowed the Rosetta community to build a base upon which tools can be constructed. This

base is the Rosetta parser and the Rosetta Object Model (ROM), a set of Java classes that allow the user access to information in a Rosetta description by semantic construct. These are already being used in commercial tools by Edaptive, among others.

The money on this contract has been well spent by the government. Rosetta now has a momentum that will, given time and care, carry it into industrial acceptance, with full support for government needs. We anticipate a longer period of acceptance for Rosetta than for VHDL (for example), but we also anticipate that Rosetta will be another government success story.

7   Lessons Learned

The first lesson learned in this effort is that adjusting for the cancellation of the final third of the contract is difficult, but not impossible. The final year of the contract was to produce a prototype simulator for a subset of Rosetta, and the lack of this tool has caused substantial problems in using Rosetta as a mature solution. It has interfered with other projects (notably the use of Rosetta on the Electronic Parts Obsolescence Initiative (EPOI), an AFRL MANTECH sponsored initiative). Not only that, but again and again we are asked for a demonstration, some sort of simulation that people can see and react to, so that Rosetta seems a real alternative. We are unable provide that given the reduction in scope and funding of the original project.

Fortunately, the cancellation of the last third of the project has not been fatal. Rosetta continues to proceed, and we hope to get more centers of Rosetta research around the world. The University of Manchester provided a fellowship for support of David Barton to get a center of research started there, and this center, while struggling to get funding, is beginning to grow.

Another lesson is that making truly innovative changes in a field is hard. Compared with Rosetta, VHDL was a baby step; RTL languages existed, and VHDL used many of their concepts. Rosetta is a quantum step in expressibility (it has to be, to meet its requirements), and convincing people that quantum steps are needed is an extremely hard task. However, a part of this lesson is that such quantum steps result in a hard core of dedicated people who can see the possibility. The contract funds have allowed the Rosetta community to develop this core. Rosetta will build and grow, and as people see its benefits it will become a valuable tool in the systems design suite of techniques.

8   Conclusions and Future Work

Thanks to the support of the Air Force Laboratories, Rosetta is defined and on its way. At least two commercial companies are actively developing Rosetta tools, and more may be doing so less publicly. University centers of research are

also being established.

This work will continue. Announcements, including a textbook, are scheduled for next years Design Automation Conference (DAC) in New Orleans. Most of all, the Rosetta community is looking for a demonstrator: something that will conclusively prove that Rosetta saves money and time to market. This has been difficult, as it requires getting industrial support in a recession; however, the effort is continuing.

However, enough has been done for a proof of concept: Rosetta meets its goals. How much it will be taken up by industry remains to be seen, but the initial work is done. As a result of this program, Rosetta exists and has a base set of tools; however, this does not assure industrial support, and lacks some basic components (a simulator and a graphical user interface (GUI)) that would help assure that support. It is possible that Rosetta will become a victim of business decisions that are not related to technical merit, but to ROI of additional needed tool development and the defined base of tools for the language. Future government support will increase the chances of success.